#1.   (" .... " + id)   → prepared Statement.

(" ... ? ; " ..")   ⇒ JDBC statement

prepared Statement may cause exceptions , it may query the whole table.
So it is not safe.   sql injection.

#2.   RowMapper.

And extract the data from result set.

#3.   By using NamedParameterJdbcTemplate, you can use colums as parameters.

#4.   @ Entity
@ Table (name = "my_wp_postmeta")
public class ~~~~ MyWpPostmeta {

        @Id
        ~~@Column~~
        @GeneratedValue
        private int meta_id;

        @ Column (name = "post_id")
        private int post_id;

        @Column (name = "meta_key")
        private String meta_key;

        @Column (name = "meta_value")
        private long meta_value;

    }

#5. For example, employeeId = employeeNum + company-id.

```java
@Embeddable.
public class Employee {
    @Column (name = "employeeNum")
    private int employeeNum;

    @Column (name = "company-id")
    private int companyId;

    public void getEmployeeId (employeeNum, companyId) {
        this.employeeNum = employeeNum;
        this.companyId = companyId;
    }
    // setter and getter ...
}
```

when using it :

```java
@EmbeddableId
private int EmployeeId;
```

#6. It is "many-to-many" relation. ✓

```java
public class Stock implements Serializable {
    private int stockId;
    private String stockCode;
    private String stockName;
    private Set<Category> categories = new HashSet<Category>

    @ManyToMany ( Fetch = FetchType.LAZY, Cascade.All)
    @JoinTable ( name = "stock- category")
    @JoinTable ( @JoinColumns = {@JoinColumn( name ="Stock_ID") }
                @InverseJoinColumn = { @JoinColumn ( name = "CATOGERY_ID") }.
    public Set<Category> getCategories () {
        return this.categories;
    }
}.
```

```java
public class Category implements Serilizable{



    @ManyToMany (fetch = Fetch. LAZY          , mappedBy="categories")
      public  Set < Stock>  get Stock(
          return  this. Stock;
        }
```

#7.   get() → load *real data*
       load() → load proxy.

#8.   Translent :   when new an object.
       persistent : data has to be the same between database and hibernate.
         { after persist() / save()
         { after get() / load()   → enter into persistent state.

       detached :   evict() / close() / clear()

       removed :   delete() / remove().

#9.
```java
        try {
            tx. begin Transaction() ;

            tx. commit() ;

             session. save (nser);
          }
        Catch ( Exception e ){
             tx. roll back();
          }
          finally {
             session. close();
          }.
```

#10. It's called hibernate locking.

we have to make sure, during the commit, ~~then~~ the data is not modified by other concurrent ongoing transaction/thread.

we can use ~~p~~ optimistic locking. (there is another ^my called pessimistic locking)

① by using version, which can be a TimeStamp.
So if it is modified, the version has been changed.

② can ~~b~~ use dynamic optimistic locking.
@ optimistic Locking
@ Dynamic

creat a column named "created_on" in database, it is ~~edt~~ TimeStamp.

#11. There are two level cache.
① first level is session level, which is in order to minimize database visit.
And it is defaul, which can't be closed.

② seccond level is sessionFactory level, which is for cross session use.
It is can be configured in XML/Annotation.

There ~~is~~ another one called Query Cache. It ~~is~~ stores query results.

#12. Query Cache stores query results.
So you don't need to visit database again and again to execute the same query.

#13. Fetch Type ⎰ LAZY → only load parent object, not with children.
⎱ EAGER → load all of them.

Cascade. All (connect) one to many
many to one

#14. ~~Both extends Repository interface.~~
CRUDRepository extends Repository interface. it returns Iterable.
JPARepository returns a list.

#15. Pagination.

extends   PagingAndSorting Repository.

#16.   ①@Enable JPA Repositories

② interface UserRepository extends JPARepository < User, Long >{
        List <User> findByFirstname (String firstname);
    }
                                    AndAgeNotNull

— 0.5

③ inject into service. (constructor dependency injection).

#17).   ~~@ControllerAdvice~~.
        @ Aspect
        public class User {
            @ After (~~execution~~(*(com. ..... UserService. get~~User~~(*){
            public String getUser (int id){
            }
            @ After (~~execution~~   execution (*(com. .... UserService. getAnotherUser(*))
            public String getAnotherUser (User n){
            }.
        }

get (*) → 就是所有 get方法.

@Before. ✓

After 不可以
因为有 extension.

#18.   @ControllerAdvice.

        @ ExceptionHandler

— 1

#19.   @ Transactional. ✓

#20.   propagation ("REQUIRED") → require a transaction
            ("REQUIRED_NEW") → require a new transaction.

        isolation (" read-commit") ✓ default. read committed.

#21.   rollback for= "CustomException" class

#22.   After you submit your updated code to code library,
        the code will be packaged to WAR.   −0.5
        Dev will upload WAR and run all the tests.
        If successed,
        If everything is correct, ~~your up~~ you might get a code analysis report.
        If something goes wrong, your previous successful WAR will be uploaded.
        This is C2CD pipeline.   continuous Integeretu...
        And the most popular tool is called Jenkins.